{EPITECH}

GOMOKU_

< BOOTSTRAP />

GOMOKU

Language: Anything working on ”the dump”

At the end of this Bootstrap, you should have a sufficient global knowledge to start working on your algorithms.
2 themes must be introduced before you are able to do this:

1. the game interface and rules
2. basic AI algorithms

Game Interface

At the end of the project, your program should be able to play against AIs of various difficulty levels. For that reason, it is necessary your program complies with the same rules as everyone else:

1. Game rules. We use the freestyle ruleset here. Every player plays a stone at his/her turn, and the game ends as soon as one has 5 stones in a row (vertical, horizontal or diagonal), or if a player cannot play in the given time.
2. Game protocol. We decided to play with the official Gomoku AI protocol

Before starting to develop your AI, **implement the mandatory part of the Gomoku AI protocol**. We actually recommend you quickly write and test a dumb AI (say, random) to make sure you comply with the interface. Protocol Documentation Intranet Mirror

Once you have a working dumb AI (once again something that is random is good enough) you can test it directly using **liskvork** (the game arbitrator that you will be using for the whole project).

You can get pre-compiled binaries of **liskvork** here.

Or if you are feeling adventurous you can compile the code yourself and perhaps modify its code to test your AI in the best way possible from the Epitech Mirror or from the official repo.

Only the mandatory commands are required for this project.

Algorithms

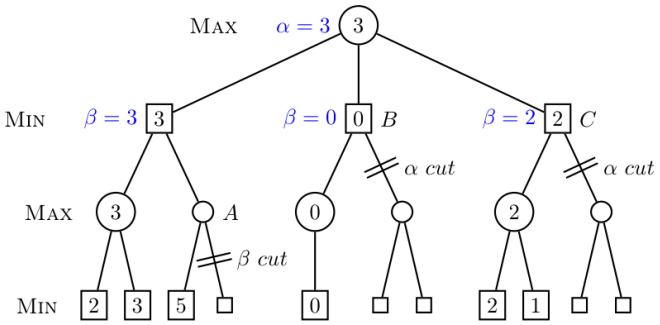
Once you have implemented the Gomoku AI protocol, you need to think about the proper AI algorithm for your bot.
Several choices seem relevant, and you have to study, understand and try.

The most obvious choice would be the **MinMax** *(not Verstappen)* algorithm (along with alpha-beta pruning and a couple of optimizations). Here you mostly have to design smart rules and control depth.

You could also try a statistical algorithm, based on the **Monte-Carlo** method for instance. These techniques mix up pretty well with the former ones.

Finally, you could go for something very powerful, and dig for **Machine Learning** based algorithms, by using NNUEs. Be careful to use only methods you are able to implement yourself, since scientific libraries are not allowed for that project (no tensorflow, scikit-learn or scipy for instance).

Of course even if you manage to implement your own NNUE you cannot use someone else's model. If you want to use that technique you will have to train your own, which is a whole art of its own.



Research all the terms you don't perfectly master, write and execute bits of code, read some papers (there are a lof them, and they are really interesting), discuss with your mates (there are also quite a few communities online to talk to about this), search for optimizations,...

By the way, any idea on how to optimize your **evaluation function**?

v 4.1